Parallel Programming of High-Performance Systems

A collaborative course of NHR@FAU and LRZ Garching

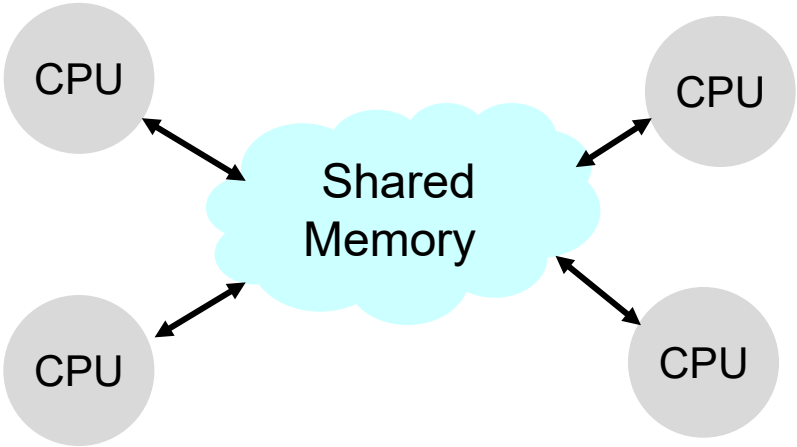
Georg Hager, Volker Weinberg, Alireza Ghasemi

Shared-Memory Computer Architecture

High Performance Computing

Shared memory

- Single address space for all processors/cores
- Cache coherent, i.e., changes in one cache will be communicated to all others for consistency

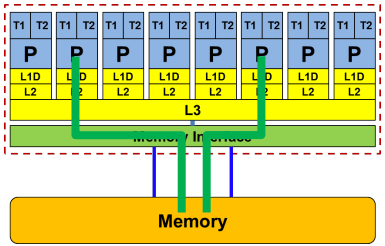


- Two basic variants: UMA and ccNUMA

UMA vs. ccNUMA

[cache-coherent]
Uniform **M**emory **A**ccess

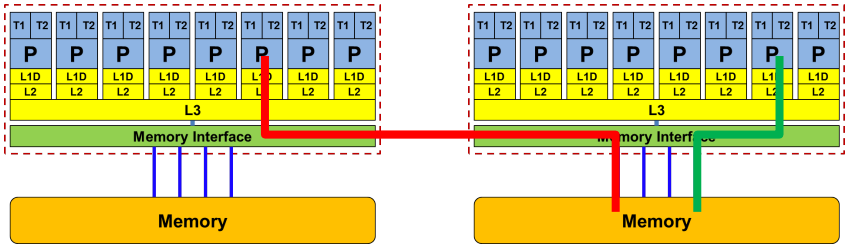
All memory accessible by all cores with same latency and bandwidth



cache-**c**oherent
Non-**U**niform **M**emory **A**ccess

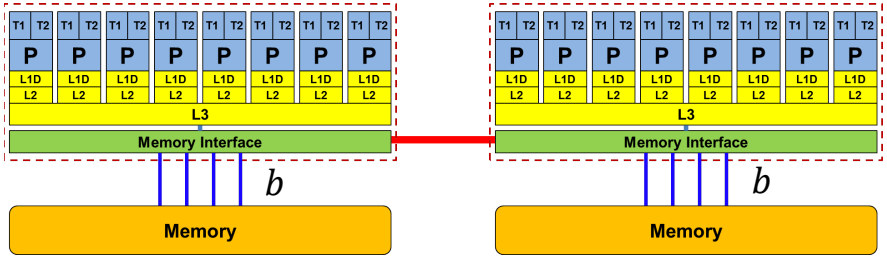
Latency and bandwidth vary depending on mutual position of core and memory

But why???



Why ccNUMA?

- Many algorithms rely on high Memory bandwidth:



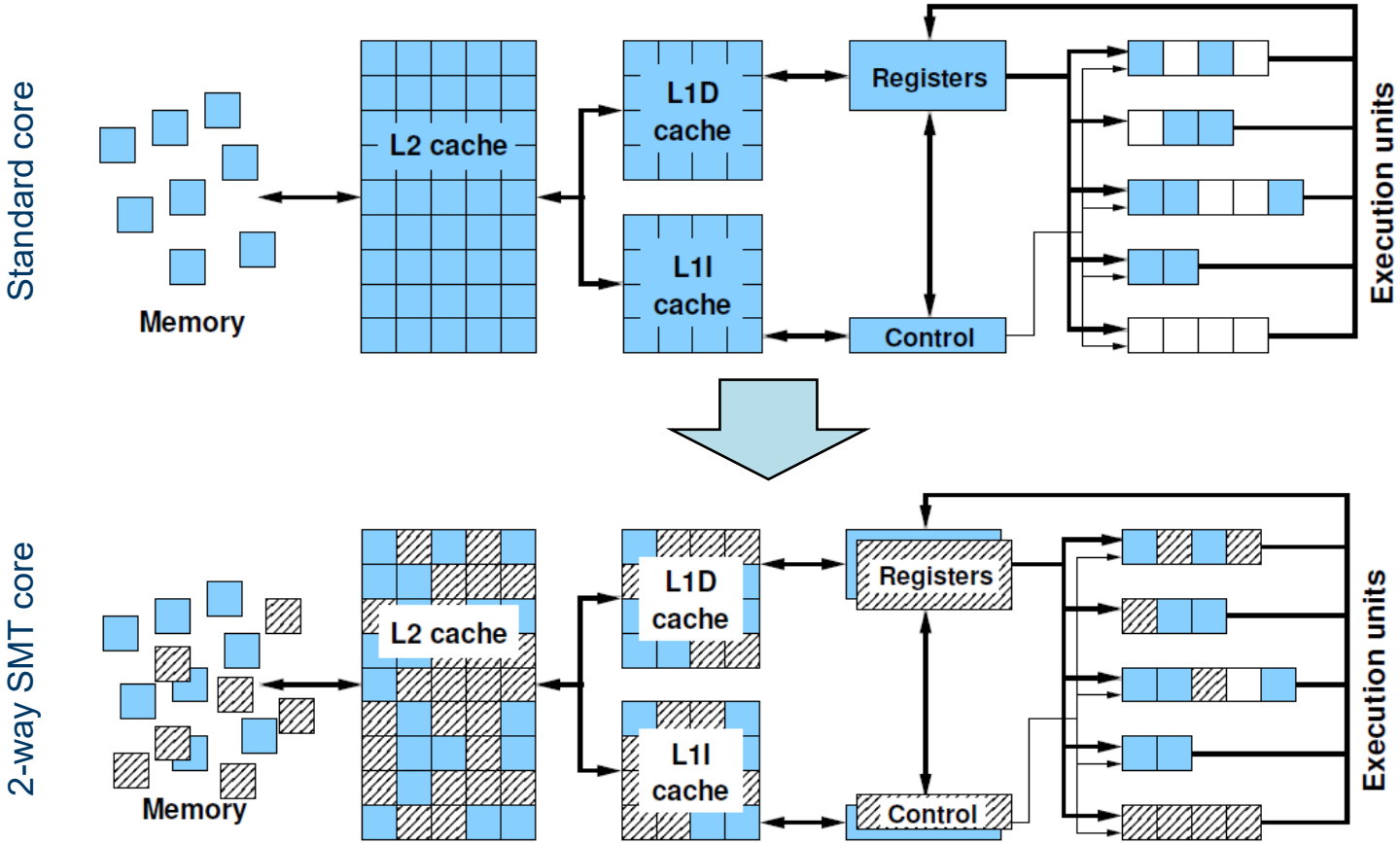
$$b = \frac{V}{T}$$

V data transferred over memory bus [byte]

T wallclock time [s]

- Advantage: Easier (cheaper) to build multiple domains with smaller bandwidth than one UMA domain with high bandwidth
- Disadvantage: Adds “topology” (non-uniformity in memory access, need to know where my threads are running)

Simultaneous multi-threading (SMT)

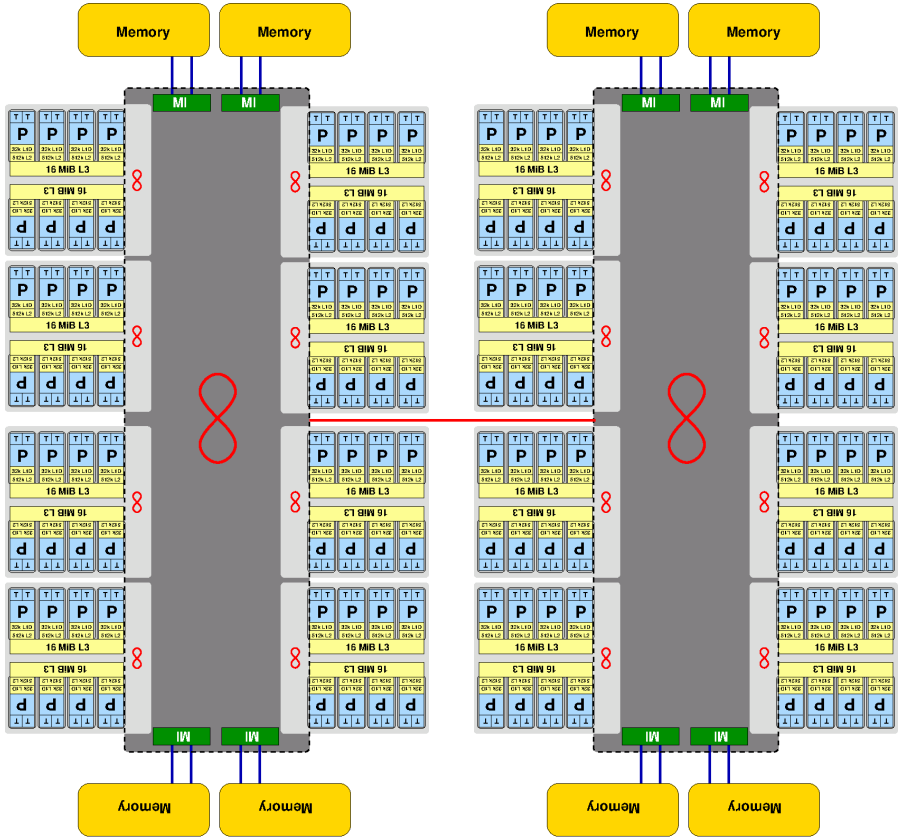


SMT benefits and caveats

- Can provide better throughput if there is parallelism in the code
  - i.e., more instructions executed per second
  - This is not automatic – code must have multiple threads/processes
  - “If in doubt, give it a try!”
- Almost all chip resources are shared among hardware threads
  - Execution units, caches, memory interface
  - Sharing these resources may prevent SMT from improving performance or even give a performance hit
- SMT introduces another layer of topology on top of it all
  - Learn how to ignore it if necessary

A modern dual-socket node

- AMD “Rome” (Zen2) dual-socket system
  - 64 cores per socket (with SMT)
  - 8 cores per die, 8 dies per socket
  - Shared L3 cache for core quadruplets (half dies)
  - AMD “Infinity Fabric” between dies and sockets
- Up to four ccNUMA domains per node
  - Configurable NPS1, NPS2, NPS4
- Two DDR4 memory channels per ccNUMA domain



NHR FAU

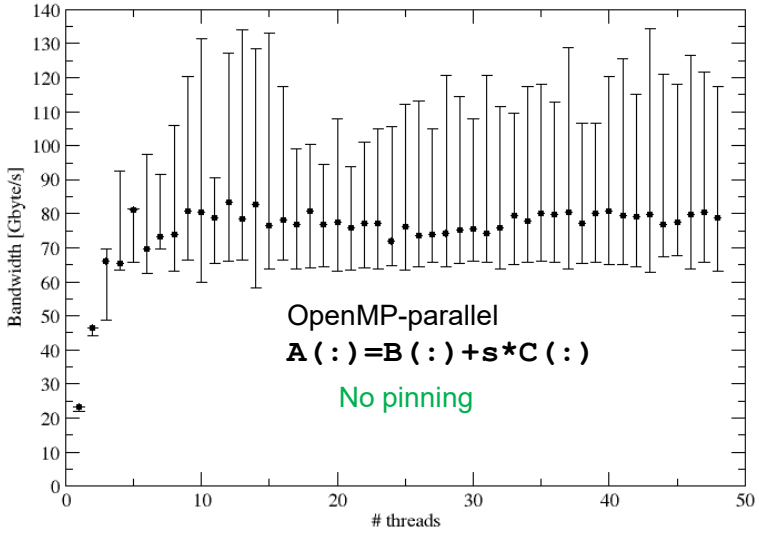
lrz

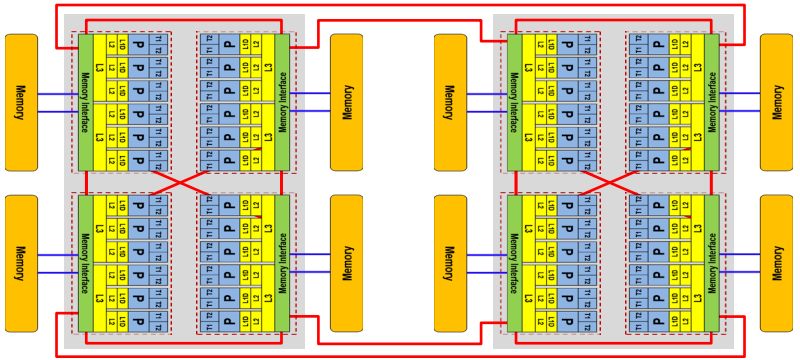
The role of thread/process affinity

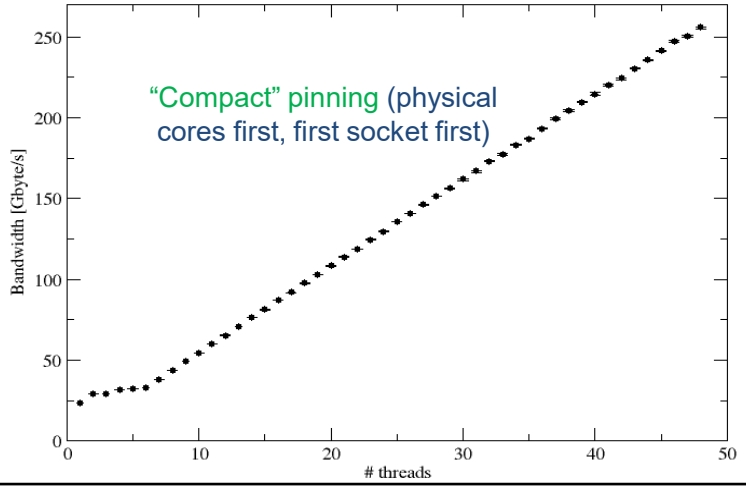
High Performance Computing

STREAM benchmark on 2x24-core AMD Zen “Naples”

Anarchy vs. thread pinning







There are several reasons for caring about affinity:

- Eliminating performance variation
- Making use of architectural features
- Avoiding resource contention

NHR FAU

lrz

Cache coherence

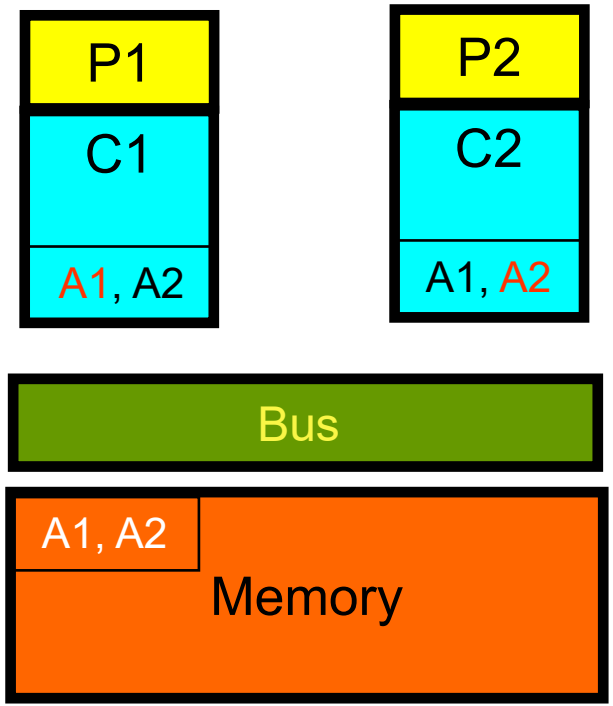
HPC High Performance Computing

Cache coherence in shared-memory computers

- Data in cache is only a copy of data in memory
  - Data is always cached in blocks (“cache lines”) of, e.g., 64 bytes
  - Multiple copies of same data on multiprocessor systems – consistency?
  - Without cache coherence, shared cache lines can become clobbered

- Cache coherence protocol keeps track of cache line (CL) status
  - Simple protocol: MESI
  - Cache line can be
    - Modified
    - Exclusive
    - Shared
    - Invalid

Without cache coherence protocol



P1	P2
`Load A1`	`Load A2`
`Write A1=0`	`Write A2=0`

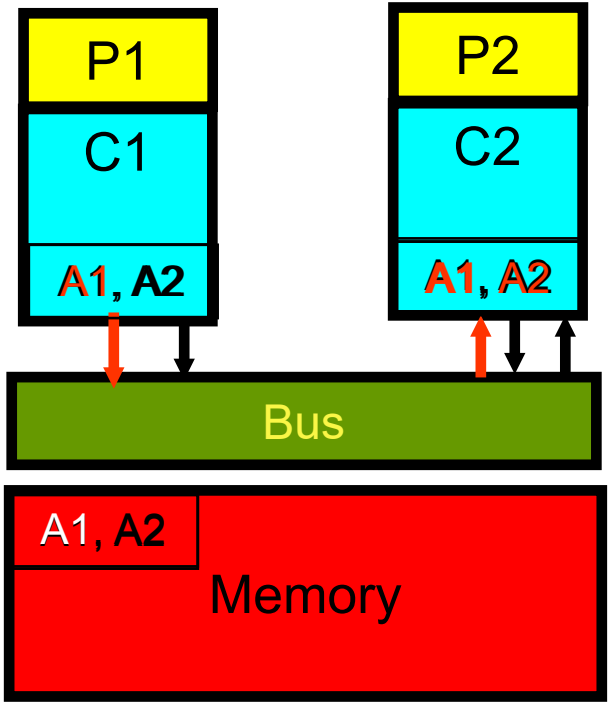
Write-back to memory leads to incoherent data

A1, A2 | A1, A2 | A1, A2

C1 & C2 entry can not be merged to:

A1, A2

With cache coherence protocol



P2

Load A1

Write A1=0:

1. Request exclusive access to CL
2. Invalidate CL in C2
3. Modify A1 in C1

Load A2

Write A2=0:

1. Request exclusive CL access
2. CL write back+ Invalidate
3. Load CL to C2
4. Modify A2 in C2

t

C2 is exclusive owner of CL

Cache coherence

- Cache coherence can cause substantial overhead
  - may reduce available bandwidth
  - “False sharing” when multiple cores modify same CL frequently
- Different implementations
  - Snoop: On modifying a CL, a CPU must broadcast its address to the whole system
  - Directory, “snoop filter”: Hardware (“network”) keeps track of which CLs are where and filters coherence traffic
- Directory-based ccNUMA can reduce pain of additional coherence traffic

- Multiple cores should never write frequently to the same cache line (“*false sharing*”)! Very bad performance may ensue.

Summary on shared-memory architecture

- Basic building block of all modern CPU-based clusters: shared-memory "compute node"
- Significant "topology" within the node
  - Simultaneous multi-threading (hyper-threading)
  - Shared/private caches
  - Memory interfaces
  - Sockets ("packages")
- Topology has important performance implications
  - Thread-core affinity (pinning) is decisive!
- Cache coherence mechanisms make programming easier
  - In general, nothing to worry about except when you have to ;-)

Parallel Programming of High-Performance Systems

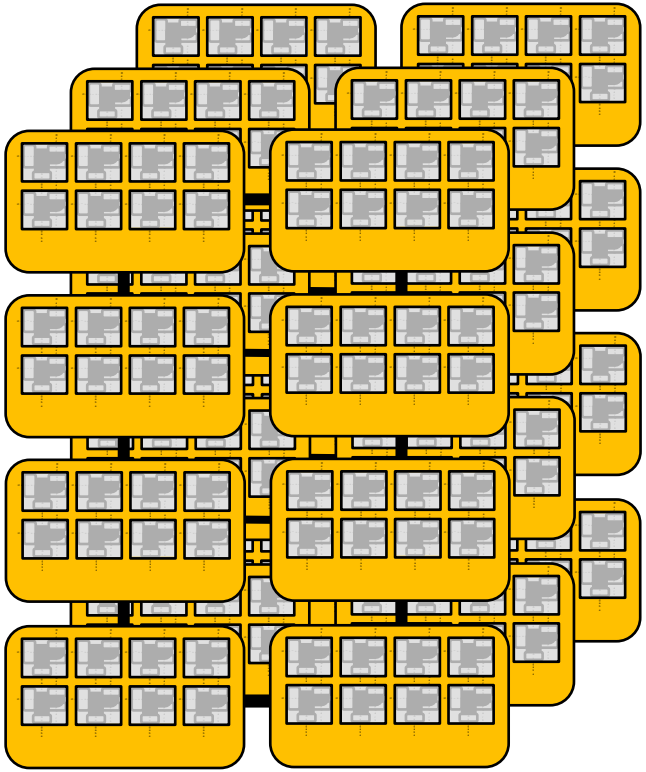
A collaborative course of NHR@FAU and LRZ Garching

Georg Hager, Volker Weinberg, Alireza Ghasemi

Distributed-Memory Computer Architecture

High Performance Computing

Distributed memory: no cache-coherent single address space



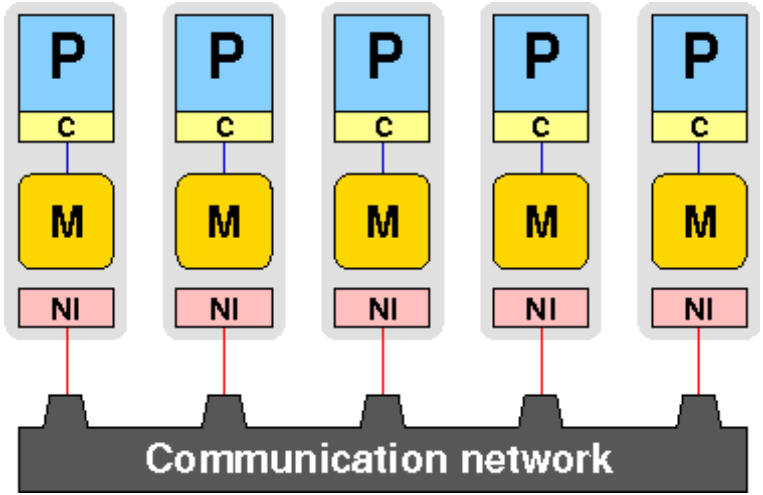
Cluster/
supercomputer

Modern supercomputers are
shared-/distributed-memory hybrids

Distributed-memory systems “back in the day”

“Pure” distributed-memory system:

- Individual processors with exclusive local memory (M) and a network interface (NI) → one “node” == one processor core
- Dedicated communication network
- Parallel program == one process per node
- Data exchange via “message passing” over the network

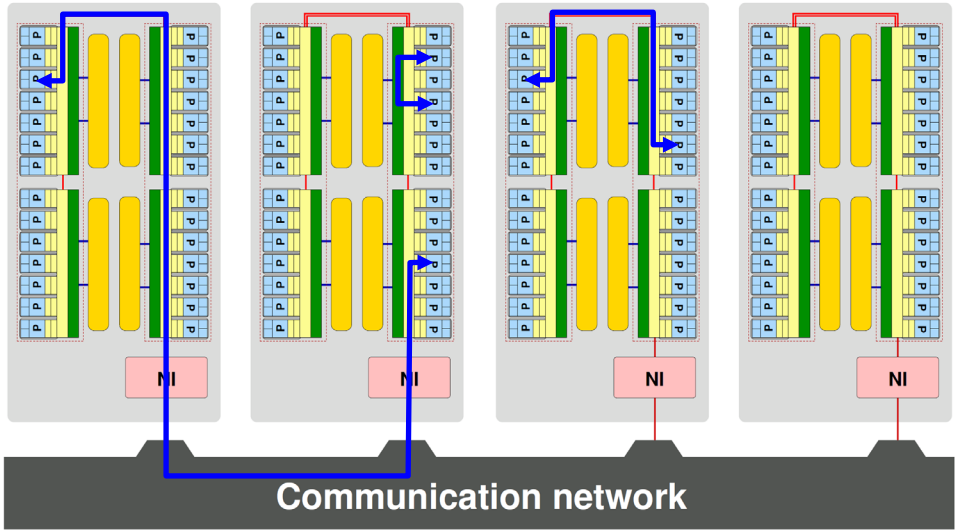


- This was a thing not so long ago…

Distributed-memory systems today

"Hybrid" distributed-/shared-memory systems

- Cluster of networked shared-memory nodes
- ccNUMA architecture per node
- Multiple cores per ccNUMA domain



- Expect strong topology effects in communication performance
  - Intra-socket, inter-socket, inter-node, all have different λ and b
  - On top: Effects from network structure

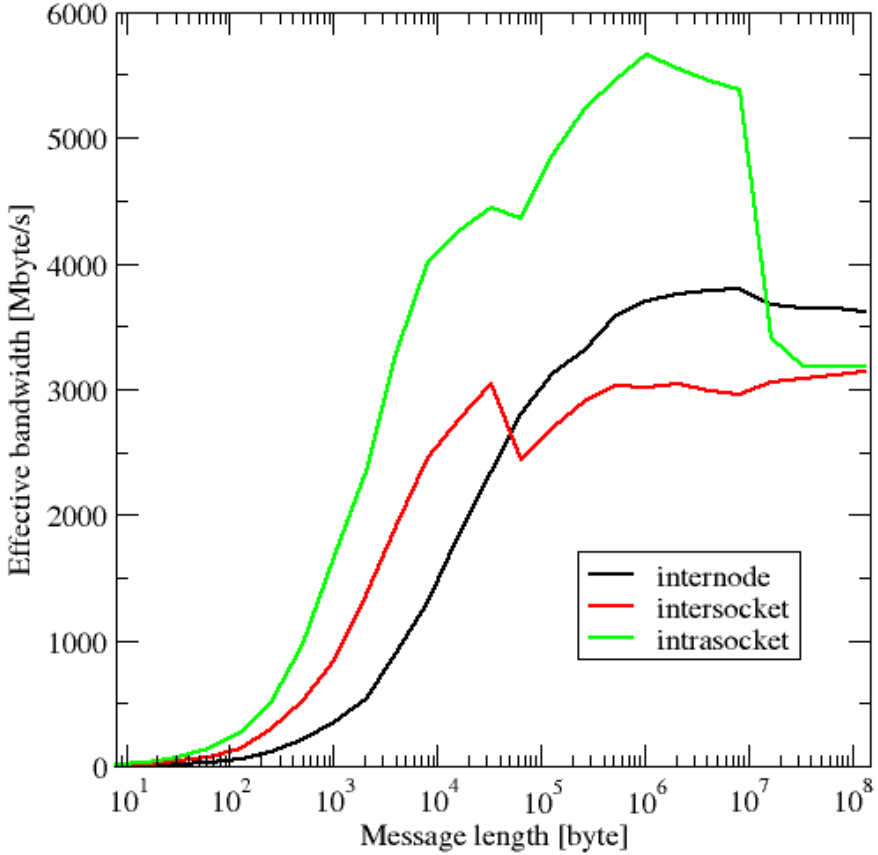
Point-to-point data transmission performance

- Simple "Hockney model" for data transfer time

$$T_{comm} = \lambda + \frac{V}{b}, \quad B_{\text{eff}} = \frac{V}{T_{comm}}$$

λ: latency, b: asymptotic BW

- Reality is more complicated
  - System topology
  - Caching effects
  - Contention effects
  - Protocol switches



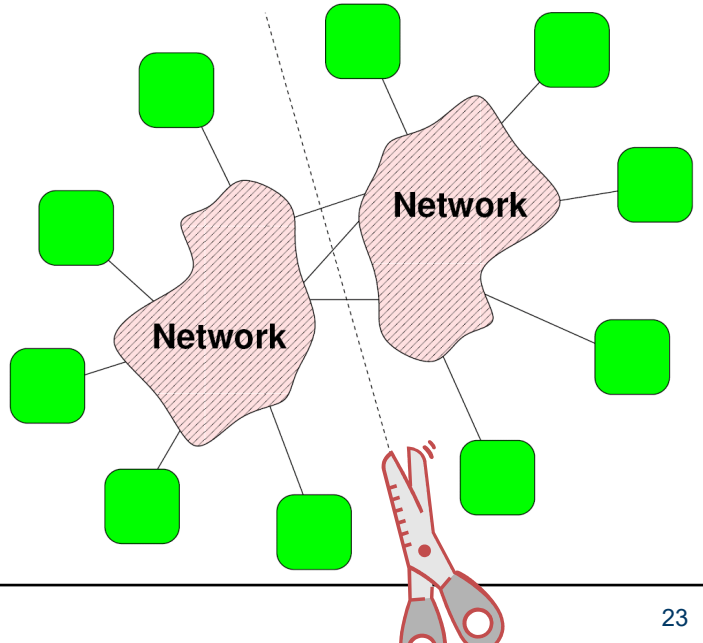
Characterizing communication networks

- Network bisection bandwidth B_b is a general metric for the data transfer "capability" of a system:

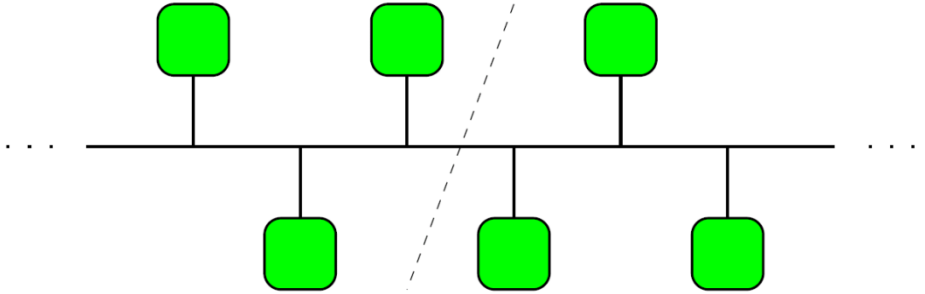
  Minimum sum of the bandwidths of all connections cut when splitting the system into two equal parts

- More meaningful metric for system scalability: bisection BW per node: B_b/N_{nodes}

- Bisection BW depends on
  - Bandwidth per link
  - Network topology



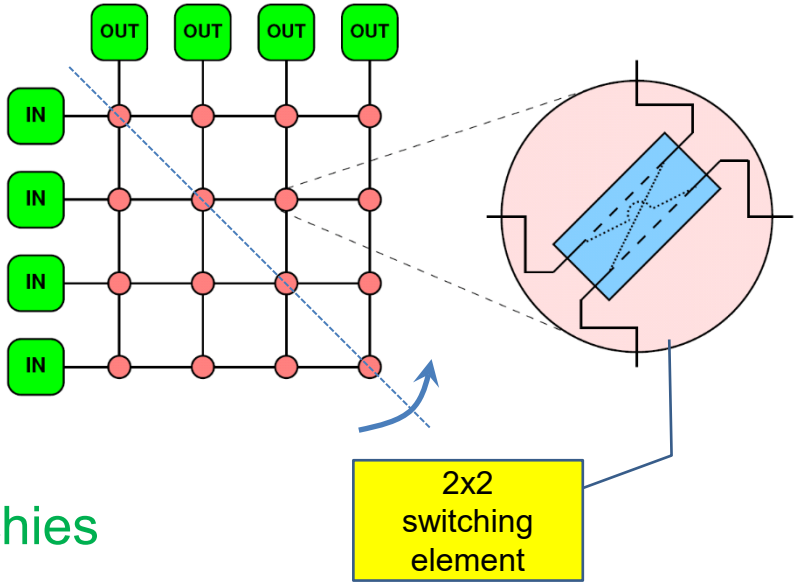
Network topologies: bus



- Bus can be used by one connection at a time
- Bandwidth is shared among all devices
- Bisection BW is constant → $B_b/N_{nodes} \sim 1/N_{nodes}$
- Examples: diagnostic buses, old Ethernet network with hubs, Wi-Fi channel
- Advantages
  - Low latency
  - Easy to implement
- Disadvantages
  - Shared bandwidth, not scalable
  - Problems with failure resiliency (one defective agent may block bus)
  - Large signal power per agent

Network topologies: non-blocking crossbar

- Non-blocking crossbar can mediate a number of connections among groups of input and output elements
- This can be used as a n-port non-blocking switch (fold at the secondary diagonal)



- Switches can be cascaded to form hierarchies (common case)
  - Allows scalable communication at high hardware/energy costs
  - Crossbars are rarely used as interconnects for large scale computers
    - NEC SX9 vector system (“IXS”)

Network topologies: switches and fat trees

- Standard clusters are built with switched networks
- Compute nodes (“devices”) are split up in groups – each group is connected to single (non-blocking crossbar-)switch (“leaf switches”)
- Leaf switches are connected with each other using an additional switch hierarchy (“spine switches”) or directly (for small configurations)
- Switched networks: “Distance” between any two devices is heterogeneous (number of “hops” in switch hierarchy)
- Diameter of network: The maximum number of hops required to connect two arbitrary devices (e.g., diameter of bus=1)

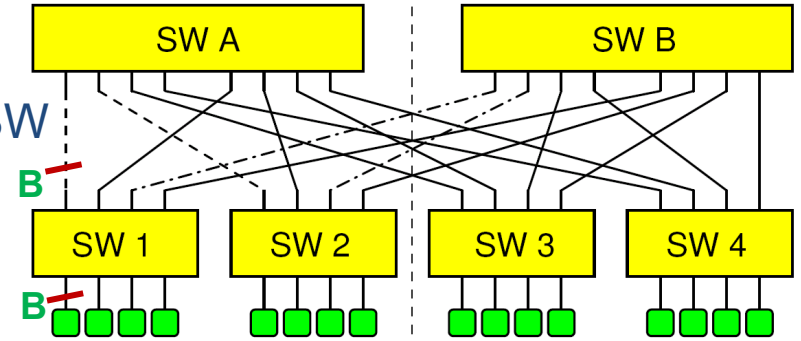
- “Perfect” world: “Fully non-blocking”, i.e. any choice of $N_{nodes}/2$ disjoint node (device) pairs can communicate at full speed

Fat-tree switch hierarchies

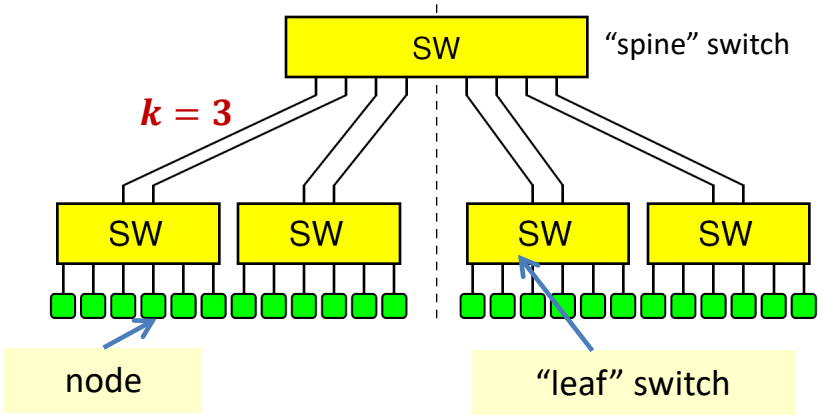
- "Fully non-blocking"
  - $N_{nodes}/2$ end-to-end con-nections with full BW

    → $B_b = B \times N_{nodes}/2$, $B_b/N_{nodes} = B/2$

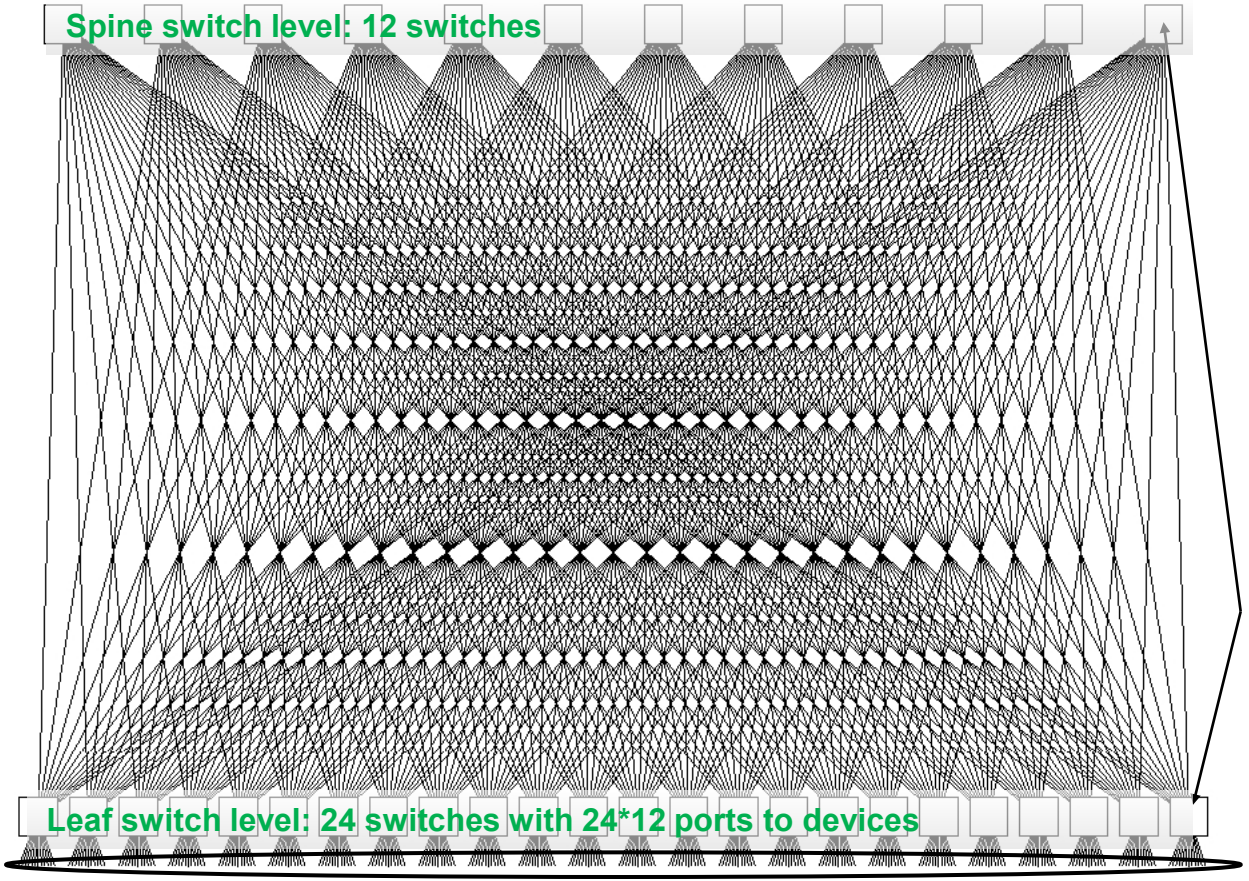
  - Sounds good, but see next slide



- "Pruned tree"
  - Spine does not support $N_{nodes}/2$ full BW end-to-end connections
  - $B_b/N_{nodes} = const. = B/(2k)$, with k = pruning factor
  - Resource management (job placement) is crucial



A “single” 288-port InfiniBand DDR switch



Examples for fat-tree networks in HPC

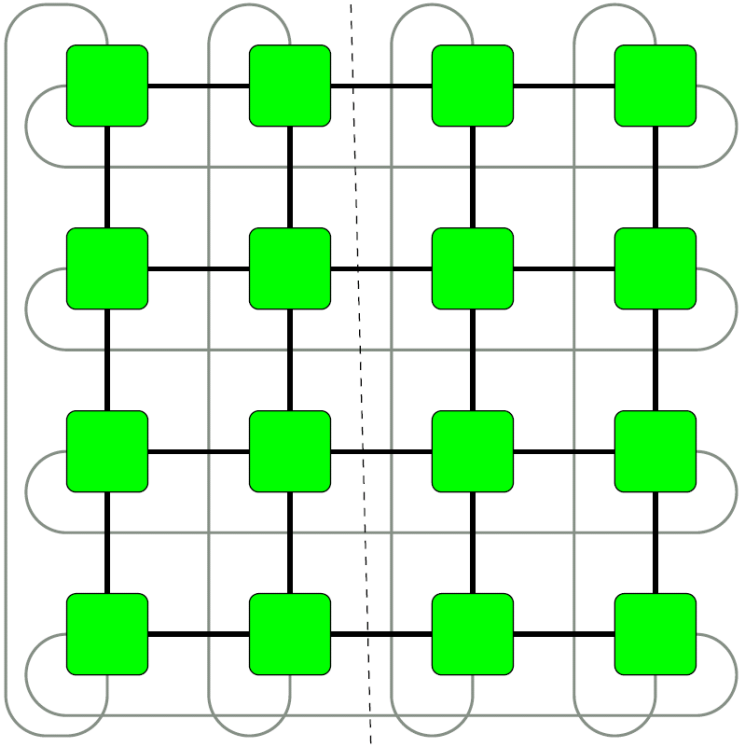
- Ethernet
  - 1, 10, 25, and 100 Gbit/s variants

- InfiniBand: Dominant high-performance “commodity” interconnect
  - DDR: 20 Gbit/s per link and direction (Building blocks: 24-port switches)
  - QDR: 40 Gbit/s per link and direction, building blocks: 36-port switches
    → “Large” 36x18=648-port switches
  - FDR-10 / FDR: 40/56 Gbit/s per link and direction
  - EDR: 100 Gbit/s per link and direction, HDR: 200 Gbit/s

- Expensive & complex to scale to very high node counts

Mesh networks

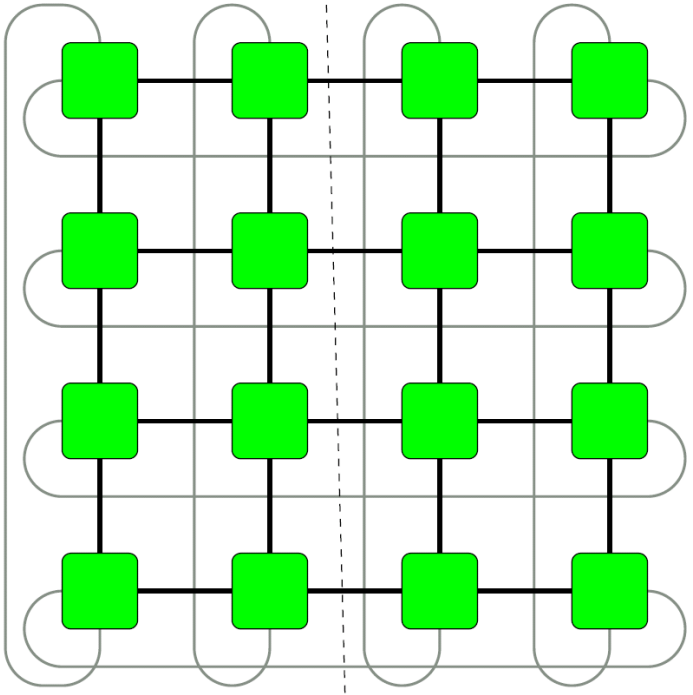
- Fat trees can become prohibitively expensive in large systems
- Compromise: Meshes
  - n-dimensional Hypercubes
  - Toruses (2D / 3D)
  - Dragonfly
  - Many others (including hybrids)



Example: 2D torus mesh

2D torus mesh

- This is not a non-blocking corossbar!
  - Intelligent resource management and routing algorithms are essential
- Direct connections only between direct neighbors
  - Each node is/has a router
- Toruses in very large systems: Cray XE/XK series, IBM Blue Gene
  - $B_b \sim N_{nodes}^{(d-1)/d}$ → $B_b/N_{nodes} \to 0$ for large N_{nodes}
  - Sounds bad, but those machines show good scaling for many codes
  - Well-defined and predictable bandwidth behavior!



HPE Slingshot (Dragonfly topology)

HPE SLINGSHOT

Dragonfly Network Architecture

- Packet-by-packet routing of unordered traffic (e.g. MPI/Lustre bulk data) optimally routed at each hop
- Adaptive routing of ordered traffic (e.g. Ethernet) Each new flow can take an optimal new path

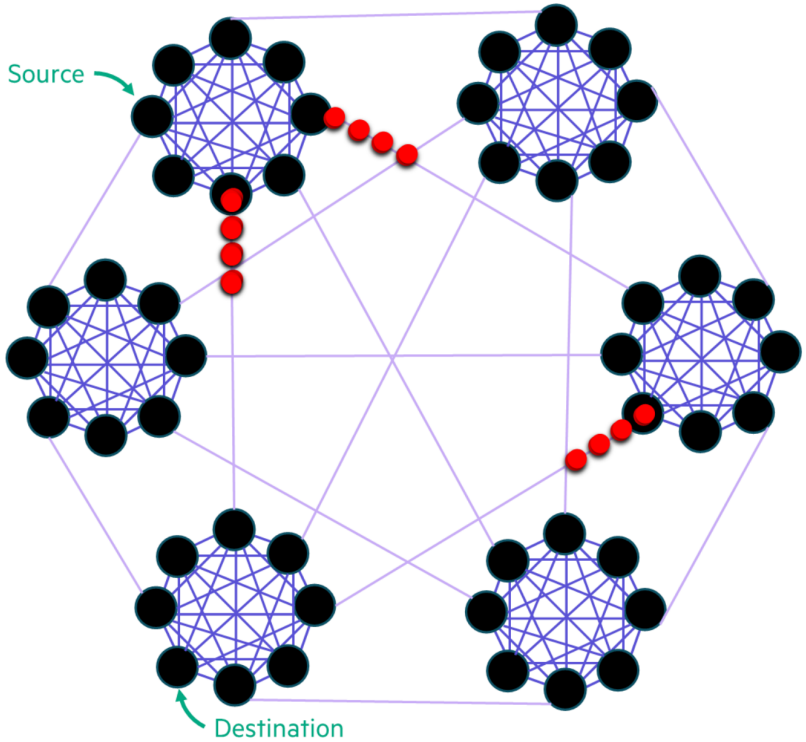
Rosetta Switch

64 port switch, 200 Gb/s

- Advanced adaptive routing
- Congestion control, QoS

Cassini NIC

- MPI hardware tag matching
- MPI progress engine
- Hardware support for one-sided operations
- Hardware support for collective operations
- 200 Gb/s



Slide by C. Simmendinger, HPE

Summary of distributed-memory architecture

- "Pure" distributed-memory parallel systems are rare
  - Hierarchical parallelism rules
- Simple latency/bandwidth model good for insights, but unrealistic
  - Protocol switches, contention
- Wide variety of network topologies available
  - Nonblocking crossbar
  - Fat tree
  - Meshes (torus, hypercube, Dragonfly, hybrids)
  - Adds more layers of topology on top of node level

- For advanced programming of hybrid hierarchical systems, see "Hybrid Programming in HPC – MPI+X" tutorial by HLRS, NHR@FAU, VSC